Beyond Modes: Building a Secure Record Protocol from a Cryptographic Sponge Permutation

Markku-Juhani O. Saarinen

Kudelski Security, Switzerland
mjos@cblnk.com

Abstract. BLINKER is a light-weight cryptographic suite and record protocol built from a single permutation. Its design is based on the Sponge construction used by the SHA-3 algorithm KECCAK. We examine the SpongeWrap authenticated encryption mode and expand its padding mechanism to offer explicit domain separation and enhanced security for our specific requirements: shared secret half-duplex keying, encryption, and a MAC-and-continue mode. We motivate these enhancements by showing that unlike legacy protocols, the resulting record protocol is secure against a two-channel synchronization attack while also having a significantly smaller implementation footprint. The design facilitates security proofs directly from a single cryptographic primitive (a single security assumption) rather than via idealization of multitude of algorithms, paddings and modes of operation. The protocol is also uniquely suitable for an autonomous or semi-autonomous hardware implementation of protocols where the secrets never leave the module, making it attractive for smart card and HSM designs.

Keywords: Lightweight Security, Sponge-based Protocols, Sponge Construction, Autonomous Hardware Encryption, Half-duplex security, BLINKER.

1 Introduction

The last decade has seen significant advances in encryption algorithm design for pervasive and low-resource platforms; PRESENT [1] (2007),Grain-128a [2, 3] (2006-2011), Hummingbird-2 [4, 5] (2009-2011), and FIDES [6] (2013) are some notable examples, each representing a different cipher design methodology; block ciphers, stream ciphers, and authenticated encryption algorithms have been proposed [7]. However, there have been few general-purpose security suite proposals that have been designed from ground up for lightweight platforms.

In this work we forgo traditional ciphers and hashes and take a fresh look at designing light-weight security protocols. We see that a single cryptographic sponge permutation can fulfill all security requirements of such a protocol, leading to a reduction of implementation footprint and facilitating straight-forward security proofs.

Our aim is to create a generic short-distance link layer security provider that can function independently from upper layer application functions. Ideally this would be realizable with autonomous hardware, without much CPU or MCU involvement.

Contributions and structure of this paper. After a brief introduction to resource-hungry legacy record protocols (Section 2), we describe the two-channel synchronization problem which affects most of them – the interwoven order of messages from two communicating parties is left unauthenticated (Section 3).

Our design avoids much of the complexity of traditional security protocols by adopting a sequential state authentication mode (Section 4) which can better meet our security and efficiency requirements while facilitating straight-forward security proofs.

In order to counter the synchronization problem and to reduce implementation footprint we adopt a half-duplex mode that utilizes a fully shared state between the two parties (Section 5).

With the term *half-duplex* we are referring to a mode of communication where two parties take turns on a single channel – the corresponding ITU-T term is "simplex circuit". This is unrelated to the "Duplexing" primitive of SPONGEWRAP.

The "rolling" shared state will not only authenticate the current message but also all previous messages and secrets sent and received during the session by both parties together with their relative order.

We then recall basic facts about Sponge-based cryptography (Section 6), popularized by the NIST SHA-3 algorithm KECCAK [8, 9] and expand its functionality to two-party encryption and authentication with domain-separating multiplex padding (Section 6.1). This also addresses MAC truncation issues of the proposed authenticated encryption mode, SPONGEWRAP and the considerations expressed by NIST [10, 11].

After a brief technical description of authentication and (re)keying flow (Section 7), we give implementation notes (Section 8), followed by Conclusions (Section 9).

2 Legacy Record and Transport Protocols

All of the standard networking security protocols - SSL3 [12], SSH2 [13, 14], TLS [15], IPSEC [16–18], PPTP [19], and wireless WPA2 [20] together with its predecessors - can be divided into two largely independent protocols: the handshake / authentication protocol and the transport / record protocol. In this work we concentrate on the latter protocol which performs encryption and authentication of bulk data. We call these collectively as "legacy record protocols".

The record transport mechanisms of these protocols require that a diverse set of binary strings are fed to various padding, wrapping, encryption and message authentication algorithms. We denote this compound operation by f_{cs} for some "ciphersuite" determined during the handshake phase of the protocol.

In addition to the plaintext P, data items required to perform authenticated encryption usually include at least the following:

S	Incremental message sequence number for MACs.
IV	Initialization vector for block ciphers.
K_e	Secret key for the symmetric encryption algorithm.
K_a	Secret key for the message authentication algorithm.

All of this state data is required to create a protected record C which contains plaintext headers, encrypted headers, encrypted payload, padding, and the MAC.

$$C = f_{\mathsf{cs}}(P, S, IV, K_e, K_a). \tag{1}$$

The inverse typically yields either the plaintext or failure and closure of the channel:

$$f_{\mathsf{cs}}^{-1}(C, S, IV, K_e, K_a) = P \; or \; \mathsf{FAIL}. \tag{2}$$

We note that this was not the original specified behavior of these legacy protocols; various error messages were specified and implemented but these have been found to act as oracles and leak secret information in cryptanalytic attacks [21–23].

Details of f_{cs} process vary depending on the particular protocol and version, but generally a header is appended to the message, followed by passes with a MAC algorithm such as HMAC [24] and an encryption algorithm (typically AES [25] in CBC [26] mode or the RC4 stream cipher [27]). In recent years the AES-GCM [28] authenticated encryption mode has also been integrated with many of these protocols, but it is not very popular in implementations. The Wireless Protected Access 802.11i (WPA / WPA2) protocol [20] requires AES in two-pass CCM [29] mode to implement its CCMP protocol and SHA-1 [30] for key derivation. Furthermore TLS-based EAP-TLS [31] authentication is recommended.

State and Algorithmic Complexity. At least two sets of data items (state) are required since these protocols view the server-to-client and client-to-server channels as entirely independent from each other. In IPSEC the two separate Security Associations (SAs) may even theoretically utilize different algorithms.

Even if we ignore various error conditions, the security of legacy record protocols depends upon the security of a large number of unrelated component designs, including: Key derivation (PRF), HMAC and its Hash, padding, the cipher and its mode of operation, and header encoding. Furthermore all data is processed at least twice – by the encryption algorithm and the MAC algorithm, independently of each other. This is why these protocols cannot be considered fully suitable for embedded and lightweight applications or fully autonomous hardware implementations.

3 Two-Party Synchronization

As previously mentioned in Section 2, two independent channels are established by legacy protocols, one from client to server ($A \rightarrow B$) and another from server to client ($B \rightarrow A$). As these security protocols are often implemented as *communication layers* (e.g. HTTPS is just HTTP over a TLS layer), typically no API interface is even available to synchronize communications between the two channels.

Example: Consider the following three transcripts

$$\begin{aligned}
T1: &\quad B \rightarrow A : M_2,\;\; A \rightarrow B : M_1,\;\; A \rightarrow B : M_3 \\
T2: &\quad A \rightarrow B : M_1,\;\; B \rightarrow A : M_2,\;\; A \rightarrow B : M_3 \\
T3: &\quad A \rightarrow B : M_1,\;\; A \rightarrow B : M_3,\;\; B \rightarrow A : M_2
\end{aligned}$$

These three transcripts have precisely the same, valid, representation on the two channels when sent over IPSEC, TLS, SSL, or SSH protocols. The same authentication codes will match.

Therefore the upper protocol layers cannot determine whether M_2 was sent spontaneously by B ($T1$) or as a response to M_1 ($T2$) or to both M_1 and M_3 ($T3$). Such ambiguity can significantly affect the interpretation of M_2 in an upper layer application such as a transaction protocol and lead to security failures.

The Synchronization Problem of Two-Channel Protocols. This illustrates a fundamental security issue; despite individual message authentication, the interwoven order of the sequence of back-and-forth messages cannot be unambiguously determined and authenticated with the legacy protocols, a fundamental requirement for reliable transactions. This is why transaction records are often authenticated on the application level as well, adding another layer of complexity.

This issue also affects basic end-user interactive security as portions of server messaging can be maliciously delayed, encouraging the user to react to partial information.

We note this issue is already partially addressed by some national or regional payment terminal standards such as [32].

4 Rethinking Privacy and Authentication

Legacy record protocols apply authentication to each message individually; authentication of an individual message does not affect others any more than the $A \rightarrow B$ channel affects $B \rightarrow A$ channel. We note that such approach is not necessary as these protocols are not generally fault tolerant and therefore require reliable rather than datagram transport.

We simplify the abstraction of Equations 1 and 2 by defining an encoding transform $\mathsf{enc}()$ that takes in a state variable S_i, plaintext P_i, and padding, outputting a new state S_{i+1} and ciphertext message C_i. The ciphertext message C_i may be longer than plaintext P_i if it contains a t-bit authentication tag, which must be checked by the recipient.

$$(S_{i+1}, C_i) = \mathsf{enc}(\ S_i, P_i, pad\ ). \tag{3}$$

The decoding function $\mathsf{dec}()$ produces the same S_{i+1} and P_i from the ciphertext and equivalent S_i and padding, synchronizing the state between sender and receiver – or resulting in a failure in case of an authentication error:

$$(S_{i+1}, P_i) = \mathsf{dec}(\ S_i, C_i, pad\ )\ \mathit{or}\ \mathsf{FAIL}. \tag{4}$$

Here the intended utility of legacy protocols' MAC and Encryption secret keys and algorithms (for encryption and message authentication), sequence numbers, and initialization vectors boils down to a singular synchronized state variable whose contents depend on absorbed keying and initialization data together with all encrypted messaging transmitted thus far. The new state S_{i+1} can be then used for transmitting an another message; this is a "MAC-and-Continue" mode.

Our main security goals are largely compatible with those laid out for Authenticated Encryption [33, 34] and Duplex Sponges in particular – proofs in [35, 36] are applicable if appropriate domain-separating padding is used. See Section 6.1 for claimed security bounds for the following security goals:

priv The ciphertext result C of $\mathsf{enc}(S, P, pad)$ must be indistinguishable from random when S is random and P may be chosen by the attacker.

auth The probability of an adversary of choosing a message C that does not result in a FAIL in $\mathsf{dec}(S, C, pad)$ without knowledge of S is bound by a function of the authentication tag size t and number of trials.

We define an additional nonstandard, informal goal which relates to solving the synchronization problem of actual two-party protocols described in Section 3. It can be viewed as a direct extension of auth from an unidirectional communications channel to bidirectional channels and multi-party protocols:

sync Each party can verify that all previous messages of the session have been correctly received and the absolute order in which messages were sent.

Our security argument for this goal is derived from the fact that an encoding exists that effectively expresses each two-party session as a single, unique hash. However, it may be possible to achieve such verifiability in a protocol which is not strictly synchronous or has more than two parties, so we leave the formal definition of sync to latter work.

Comparison with Legacy Protocols. Our requirements are stronger than those commonly expected from security protocols; for example all protocols of Section 2 are easily identifiable, a concern in the operation of Firewalls and Intrusion Detection Systems (IDS) which try to profile and filter various protocols being used in a network.

Our design tries to avoid visible unencrypted sequence numbers and paddings that would allow trivial protocol and protocol version identification as it is very difficult to block something you cannot create an IDS signature for. [1]

The third, informal requirement sync appears to be new and is not met by current protocols as shown in Section 3. Here we are trying to address a real-world security concern rather than adding a vehicle for theoretical research.

We find that with Sponge approach we do not have to over-simplify our protocol when modeling it for security proofs. In analysis of a typical real world protocol, one is faced with a combinatorial explosion of interplay between details such as: crypto algorithms, message formatting and padding, modes of operation, hash constructions, MAC constructions, error codes, and key derivation. Such complexity is the main reason why "provably secure" protocols often fail in practice; the protocols have been be severely simplified and idealized for analysis.

During the 10-15 years since the protocols of Section 2 largely took their present form, a large number number of security proofs, counter-proofs and attacks have been presented, starting with [38–40] and [23, 41–44] representing some of the more recent work.

[1] Our BLINKER implementation has its origins in the stealthy communications mechanisms of an Academic RAT tool [37]. Here a HTTP port 80 channel was used and hence our traffic could not be "picked up" amongst other random things that are transmitted during web surfing.

5 Half-duplex Security Protocols with a Shared State

In BLINKER, we implement communications security for the shared channel using a single, synchronized state S_i for both directions, saving resources and 50% of state memory in the implementation. A domain separation padding mechanism distinguishes between the two communicating parties as well as data input types. Figure 1 shows an interchange of three messages with a synchronized state.

From security viewpoint, this setup has the advantage that the entire interchange or "conversation" is continually authenticated as the evolving state includes full contents of messages from both parties and the order they were sent. The security proofs interpret the state S_i as equivalent to a cryptographic hash of a full transcript of the session up to message or input i; this is achieved with specific padding.

Asymmetrically Signed Sessions and Transactions. The entire session up to point i can be cryptographically validated by signing a hash "squeezed" from the state S_i. Even if the initial session authentication is based on digital signatures, as is often the case with legacy protocols, this does not mean that the session is signed. Without Alice's signature of the protocol transcript, Bob (who also knows all symmetric authentication and encryption secrets) can easily forge a session transcript. It is rather difficult to sign a session with a protocol such as TLS, SSH2, or IPSEC since application-level hashing and processing is required. With a BLINKER-type protocol such final authentication is relatively easy to implement, an excellent feature for transaction protocols.

Real-Life Prevalence of Half-Duplex Links. Half-duplex links may seem rare to a software developer due to the widespread use of the socket programming paradigm. This illusion is often achieved by time-division duplexing (TDD). However, half-duplex is physically prevalent on sensor networks, IoT and last-hop radio links – Bluetooth and IEEE 802.15.4 ZigBee being two notable examples.

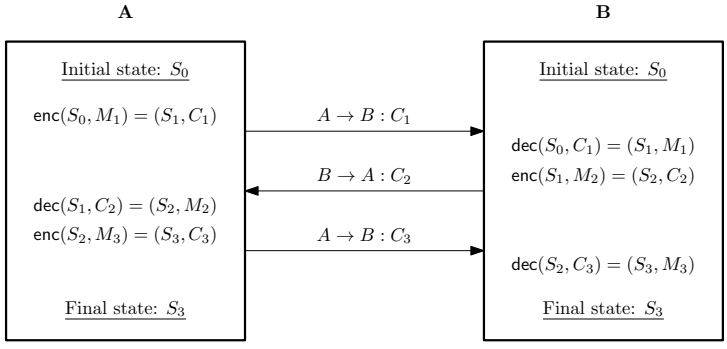

Fig. 1. Simplified interchange of three messages whose plaintext equivalents are $A \to B : M_1$, $B \to A : M_2$, $A \to B : M_3$, utilizing a synchronized secret state variables S_i. The order of messages cannot be modified and hence this exchange is sync - secure.

Half-duplex links can be established wirelessly with unpaired frequencies (same frequency in both directions), a typical scenario in light-weight time-divide communications, our specific targets. An another example are embedded twisted-wire serial links.

We note that in addition to wireless last-hop transports, most RFID, Smart Card, and industrial control (MODBUS) communications are implemented under a query-response model and are therefore effectively half-duplex [45–47].

6 Extending the Sponge Construction

Sponge constructions generally consist of a state $S = (S^r \mid\mid S^c)$ which has $b = r + c$ bits and a b-bit keyless cryptographic permutation π. The S^r component of the state has r "rate" bits which interact with the input and the internal S^c component has c private "capacity" bits.

These components, together with suitable padding and operating rules can be used to build provable Sponge-based hashes [48], Tree Hashes [49], Message Authentication Codes (MACs) [50], Authenticated Encryption (AE) algorithms [35], and pseudorandom extractors (PRFs and PRNGs) [51].

Absorbing and Squeezing. We recall the basic Sponge hash [48] concepts of "absorbing" and "squeezing" which intuitively correspond to insertion and extraction of data to or from the sponge. Let S_i and S_{i+1} be b-bit input and output states. For absorption of padded data blocks M_i (of r bits each) we iterate:

$$S_{i+1} = \pi(\ S_i^r \oplus M_i \mid\mid S_i^c\ ). \tag{5}$$

This stage is followed by squeezing out the hash $H = H(M)$ by consecutive iterations of:

$$\begin{aligned} H &= H \mid\mid S_i^r \\ S_{i+1} &= \pi(S_i). \end{aligned} \tag{6}$$

These constructions may be transformed into a keyed MAC by considering the state S_i as secret (keyed) [50]. Keying is then equivalent to initial absorption of keying material before the payload data. MAC is squeezed out exactly like a hash.

Duplexing. A further development is the Duplex construction [35] which allows us to encrypt and decrypt data while also producing a MAC in the end with a single pass.

The state is first initialized by inserting secret keying material and non-secret randomization data to the state via the absorption mechanism of Equation 5. To encrypt plaintext blocks P_i to ciphertext blocks C_i we iterate:

$$\begin{aligned} C_i &= S_i^r \oplus P_i \\ S_{i+1} &= \pi(C_i \mid\mid S_i^c). \end{aligned} \tag{7}$$

The effect on the state is the same as that of Equation 5. The inverse – decryption operation – is almost equivalent to encryption, which in itself has significant implementation advantages:

$$\begin{aligned} P_i &= S_i^r \oplus C_i \\ S_{i+1} &= \pi(C_i \mid\mid S_i^c). \end{aligned} \tag{8}$$

After encryption or decryption, a message authentication code for the message may be squeezed out as in Equation 6 and verified. To simplify exposition, we have left some key details regarding padding. We will come back to these in Section 6.1.

MAC-and-Continue. There is really no need to constrain the iteration to a single message. With appropriate domain-separating padding the security proofs allow the sponge states to be used for any number of consecutive authenticated messages ("MAC-and-Continue") without the need for sequence numbers, and re-keying. This is one of the main observations which led to the present work and greatly reduces the latency of implementation as "initialization rounds" are not required for each message. This was also proposed as part of the original SPONGEWRAP construction.

6.1 Multiplex Padding

The SPONGEWRAP [35] and MONKEYDUPLEX [36] padding rules offer concrete Sponge-based methods for performing authenticated encryption. Recent work on implementation of SPONGEWRAP and its variants on low-resource platforms is reported in [7].

The requirements laid out in [35] for the padding rule are that they are reversible, non-empty and that the last block is non-zero. The padding rule in KECCAK is that a single 1 bit is added after the last bit of the message and also at the end of the input block.

In the Duplex construction of SPONGEWRAP additional padding is included for each input block; a secondary information bit called *frame bit* is used for domain separation. SAKURA [49] uses additional frame bits to facilitate tree hashing. It is essential that the various bits of information such as the key, authenticated data, and authenticated ciphertext can be exactly "decoded" from the Sponge input to avoid trivial padding collisions. We use a more explicit padding mechanism but the following priv and auth bounds proven in [35] (Section 5.2 on Page 332) and [50] also hold for enc():

Theorem 1 (Theorem 1 from [35]). *The* SPONGEWRAP *and BLINKER authenticated encryption modes satisfy the following privacy and authentication security bounds:*

$$\mathrm{Adv}^{\mathsf{priv}}_{\mathsf{enc}}(\mathcal{A}) < q2^{-k} + \frac{N(N+1)}{2^{c+1}} \tag{9}$$

$$\mathrm{Adv}^{\mathsf{auth}}_{\mathsf{enc}}(\mathcal{A}) < q2^{-k} + 2^{-t} + \frac{N(N+1)}{2^{c+1}} \tag{10}$$

against any single adversary $\mathcal{A}$ if $K \xleftarrow{\$} \{0,1\}^k$, tags of $l \geq t$ bits are used, π is a randomly chosen permutation, q is the number of queries and N is the number of times π is called.

Note that even the Squeezing phase can utilize padding to mark the size of desired output (as we do in Section 6.2). In KECCAK and SPONGEWRAP a convention has been adopted to have a null S_r input to π during squeezing in order to separate it from other phases (hence the requirement that padding rule does not produce null blocks). However this may lead to problems in some applications where the MAC length is not clear.

Context collision in KECCAK and SPONGEWRAP. There is no indicator for MAC length in SPONGEWRAP construction – output is simply truncated. If the sender and recipient have a different idea about the length, there is no way to detect truncation of the MAC. Different length-variants of KECCAK give different outputs for the same data simply because different data rates r are used and this affects the placing of the final padding bit. Earlier members of the SHA standard avoid this issue by having different IV values depending on the desired output length [30].

6.2 Multiplexing the Sponge

Our new padding rule is called Multiplex. Input and output blocks, encrypted and authenticated data, keys, and nonces are all different input domains and must be encoded unambiguously as Sponge inputs. Rather than using frame bits per block for domain separation as in SPONGEWRAP, the data domains are explicitly encoded. This allows many more data types to be entered into the sponge as well and clearer domain separation between them. It is essential in a shared-state two-party protocol that the originating party of the block (Alice or Bob) is also used to mark domain separation between the two.

We retain one d-bit word D in S^c for domain separation; $S^c = (S^d \mid\mid S^{c'})$ with $c' = c - d$. The iteration for arbitrary absorption, squeezing, and encryption is now:

$$S_{i+1} = \pi(\ S_i^r \oplus M_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ). \tag{11}$$

For decryption we have the following update function:

$$S_{i+1} = \pi(\ C_i \mid\mid S_i^d \oplus D_i \mid\mid S_i^{c'}\ ). \tag{12}$$

In our implementation $d = 16$ bits. Table 1 gives a description of padding mask word bits (which may be OR'ed together). Message blocks are always padded with a single "1" bit and by zeros to fill r bits, followed by the multiplex padding word. If full r bits are used in a block, the padding bit is the bit 0 in the multiplex word.

The effective information theoretic capacity is reduced by the Multiplex construction to no more than $c - 3$ rather than $c' = c - d$ if tree functions are not used.

Unlike message data, the domain separation word is always XORed with the S state bits on all operations (Equations 11 and 12). Apart from few options, the domains follow each other in application-specific predetermined order and hence two bits of entropy is sufficient to mark that separation between block types in our protocol. In addition there is a padding bit that may be located in the domain separation word if the input block is full (bit 0).

Therefore the effective c for values bounds of Theorem 1 need to be modified only by 3 bits when multiplex padding is used. We do this in order to remove the requirement for additional message padding buffers and also to follow Horton's Principle [40, 52], *"Authenticate what is being meant, not what is being said."*

The separation of the domain mask word from main input allows later expansions of functionality without breaking interface designs; for example we may adopt tree-based hashing - and by extension, tree MACs and encryption - by utilizing bits 14 and 15 of D_i for this purpose rather than adding more frame bits as in SAKURA [49]. If tree structure is used, the capacity should be reduced to $c - 4$ for security analysis. Furthermore, increasing $d > 16$ will not break existing implementations.

Since the protocol exchange can be unambiguously decoded from the sponge input $(M_1 \,||\, D_1) \,||\, (M_2 \,||\, D_2) \,||\, ..$, and we do not reset the state between messages, the proofs of Theorem 1 [35, 50] seem to apply to the protocol as a whole as well as individual messages. If one can forge an individual message authentication code or (by induction) a multi-message exchange, one can also break the Sponge in a SHA-3 - type hash construction. However, we leave the formalization and proof machinery of our informal sync goal for latter work.

Padding while Squeezing. In the squeezing phases of our construction the (inputless) output blocks are virtually padded as if $M_i = 0^r$ in Equation 11. If $s < r$ bits of the block is begin squeezed out, a single "1" bit is XORed at state S after the location of last output bit; $M_i = 0^s \,||\, 1 \,||\, 0^{r-s-1}$. This resolves the SPONGEWRAP context collision described in Section 6.1 since at least the last output block will differ for different output sizes.

We acknowledge that the solution is perhaps not ideal if the extracted hash is longer than the block size; two hashes of different size from the same message are equivalent except for the final blocks.

6.3 Sourcing π

BLINKER was originally designed together with the CBEAM algorithm [53] for integrated use in low-resource and small-footprint applications.

However, the choice of π is arbitrary if it satisfies the required security properties. KECCAK is a strong candidate as it has been selected as the NIST SHA-3 algorithm [9, 8, 54], albeit its 1600-bit state is often seen as too large for low-resource platforms and short messages. However, there are nonstandard reduced-state variants KECCAK-$f[b]$ where $b = 25 \times 2^l$ for $1 \leq l \leq 6$.

Other candidates as π donors include PHOTON [55]. QUARK [56, 57], and SPONGENT [58]. Each of these can be used to construct extremely lightweight protocols based on our Multiplex / BLINKER construction.

Note that some clearly "non-hermetic" Sponge permutations such as FIDES [6] are probably not secure enough. It may be possible to be somewhat flexible in this requirement as we assume a randomized session S, as is done in the MONKEYDUPLEX [36] construct.

Table 1. Proposed bits used in the Multiplex Padding Word which is XORed with the state. Depending on protocol state and the intended usage of message block, multiple bits are set simultaneously.

Bit	Mask	When set
0	`0x0001`	This is a full input or output block (r bits).
1	`0x0002`	This is the final block of this data element.
4	`0x0004`	Block is an input to sponge ("absorption").
3	`0x0008`	Block is output from sponge ("squeezing").
4	`0x0010`	Associated Authenticated Data input.
5	`0x0020`	Secret key block.
6	`0x0040`	Nonce input block.
7	`0x0080`	Encryption / Decryption block.
8	`0x0100`	Hash block.
9	`0x0200`	Keyed Message Authentication Code (MAC) output block.
10	`0x0400`	Block for state storage or reloading.
11	`0x0800`	Pseudo Random Number Generator (PRNG) block.
12	`0x1000`	Originating from Alice (client / slave).
13	`0x2000`	Originating from Bob (server / master).
14	`0x4000`	Tree chaining Node.
15	`0x8000`	Tree final Node.

Comparison with AES-based Protocols. For most of these π permutations the working memory required to implement the entire two-way BLINKER protocol is only slightly more than b bits for the state. It is difficult if not impossible to implement AES in any reasonable authenticated mode of operation with such a small amount of memory in a two-party protocol as additional storage is required for two round / nonce counters, authenticators, and round keys.

7 Basic Shared Secret Authentication and Record Protocol Flow

We assume that the shared secret K is simply stored by both parties; however it may be derived with a lightweight asymmetric key exchange method such as Curve25519 [59]. K may also be combined from passwords or composed in other ways.

We use the shorthand $\mathsf{enc}(state, input, pad)$ in the following for encoding operations. Corresponding synchronized decoding may result in FAIL and immediate closure of channel. We do not explicitly describe these operations; see Sections 4 and 5. However, in order to clarify exposition, we are "writing out" the authentication tag generation phases.

We first absorb the identities I_a and I_b of Alice and Bob into the state. Note that it may not be necessary to transmit the messages M_1 and M_2 if the identities are self-evident. The key is never transmitted but simply mixed with the state. Let S_0 be some initialization value.

$$(S_1, M_1) = \mathsf{enc}(S_0, I_a, \texttt{0x108C}) \mid A \rightarrow B : M_1$$
$$(S_2, M_2) = \mathsf{enc}(S_1, I_b, \texttt{0x208C}) \mid B \rightarrow A : M_2$$
$$S_3 = \mathsf{enc}(S_2, K, \texttt{0x3024}) \mid$$

Two random nonces R_a and R_b are required for challenge-response authentication and to make the session unique.

$$(S_4, M_3) = \mathsf{enc}(S_3, R_a, \texttt{0x10CC}) \mid A \rightarrow B : M_3$$
$$(S_5, M_4) = \mathsf{enc}(S_4, R_b, \texttt{0x20CC}) \mid B \rightarrow A : M_4$$

We may now perform mutual authentication with tags of t bits:

$$(S_6, M_5) = \mathsf{enc}(S_5, 0^t, \texttt{0x1208}) \mid A \rightarrow B : M_5$$
$$(S_7, M_6) = \mathsf{enc}(S_6, 0^t, \texttt{0x2208}) \mid B \rightarrow A : M_6$$

Checking M_5 and M_6 completes mutual authentication. By an inductive process we see that the session secret S_7 is now dependent upon randomizers from both parties and the original shared secret is not leaked if the Sponge satisfies our security axioms.

After this, plaintexts P_a (for $A \rightarrow B$) and P_b (for $B \rightarrow A$) can be encrypted, transmitted and authenticated by repeathing the following exchange:

$$(S_{i+1}, M_a) = \mathsf{enc}(S_i, P_a, \texttt{0x108C}) \mid A \rightarrow B : M_a$$
$$(S_{i+2}, T_a) = \mathsf{enc}(S_{i+1}, 0^t, \texttt{0x1208}) \mid A \rightarrow B : T_a$$
$$(S_{i+3}, M_b) = \mathsf{enc}(S_{i+2}, P_b, \texttt{0x208C}) \mid B \rightarrow A : M_b$$
$$(S_{i+4}, T_b) = \mathsf{enc}(S_{i+3}, 0^t, \texttt{0x2208}) \mid B \rightarrow A : T_b$$

Due to explicit padding it is easy to show that the entire message flow is authenticated if appropriate checks are made.

8 Implementation Notes

We have already fielded BLINKER in a tiny security application that communicates with a server over a HTTP 1.1 stay-alive link [37]. Such a link is essentially half-duplex as messages are sent and received over HTTP POST method within a single stay-alive TCP session. On the target platform this proved to be an ideal method for communicating with a server over the Internet; SSL is essentially unimplementable on the target platform. The same is true for many low-end embedded devices that have only rudimentary TCP stacks or use some non-TCP protocol for the initial hop.

Figure 2 shows a simplified interface for a module that implements BLINKER in hardware. The mode of operation is determined by the domain separation padding word PADDING IN (as specified in Table 1) together with the SEND / RECEIVE signal that distinguishes between encryption and decryption, MAC generation and verification. It is noteworthy that the S^c secret state bits never have to leave the module and can be isolated from CPU with the interface provided.

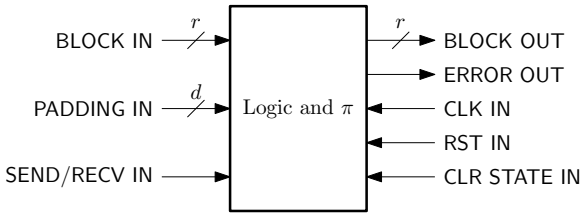

Fig. 2. A simplified interface architecture for a semi-autonomous hardware component implementing BLINKER.

9 Conclusions

We have described the use of Sponge permutations to build complete lightweight two-way communications links (record protocols). In terms of embedded RAM and ROM our design has much smaller implementation footprint when compared to traditional approaches. Furthermore the "half-duplex" design is naturally suited for these platforms and is resistant to synchronization flaws; each authentication tag essentially authenticates the entire session up to that point.

In a hardware implementation the session secrets never have to leave (and cannot leave) a specific hardware component, making the design attractive in HSM and smart card applications. Such separation is very difficult (and costly) to achieve with SSL and other legacy protocols which generally require CPU/MCU interaction to create encryption and authentication keys from session secrets.

Our design is especially suitable for last-lap and autonomous hardware communications, such as those with sensors, Radio Frequency Identification (RFID) and Near Field Communication (NFC) systems, smart cards, and Internet-of-Things applications.

Acknowledgements. The author wishes to thank Kudelski Security, University of Haifa, and Nanyang Technological University for supporting his work. Program Committee members of CT-RSA 2014 provided invaluable suggestions for improving the quality of this paper.

References

1. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In Paillier, P., Verbauwhede, I., eds.: CHES 2007. Volume 4727 of LNCS., Springer (2007) 450–466
2. Hell, M., Johansson, T., Meier, W.: Grain - a stream cipher for constrained environments. International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems **2**(1) (2006) 86–93
3. gren, M.A., Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of Grain-128 with optional authentication. International Journal of Wireless and Mobile Computing **5**(1) (2011) 48–59

4. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Ultra-lightweight cryptography for low-cost RFID tags: Hummingbird algorithm and protocol. Technical Report CACR-2009-29, University of Waterloo (2009)
5. Engels, D., Saarinen, M.J.O., Schweitzer, P., Smith, E.M.: The Hummingbird-2 lightweight authenticated encryption algorithm. In Juels, A., Paar, C., eds.: RFIDSec '11. Volume 7055 of LNCS., Springer (2011) 19–31
6. Bilgin, B., Bogdadov, A., Knežević, M., Mendel, F., Wang, Q.: Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In Bertoni, G., Coron, J.S., eds.: CHES 2013. Volume 8086 of LNCS., Springer (2013) 142–158
7. Yalçın, T., Kavun, E.B.: On the implementation aspects of sponge-based authenticated encryption for pervasive devices. In Mangard, S., ed.: CARDIS 2012. Volume 7771 of LNCS., Springer (2013) 141–157
8. NIST: NIST selects winner of secure hash algorithm (SHA-3) competition. NIST Tech Beat Newsletter (2 October 2012)
9. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference, version 3.0. NIST SHA3 Submission Document (January 2011)
10. Kelsey, J.: SHA3: Where we've been, where we're going. Talk Given at RSA Security Conference USA 2013 (February 2013)
11. Kelsey, J.: SHA3: Past, present, and future. Invited Talk Given at CHES 2013 (August 2013)
12. Freier, A., Karlton, P., Kocher, P.: The secure sockets layer (SSL) protocol version 3.0. IETF RFC 6101 (Historic) (August 2011)
13. Ylönen, T., Lonvick, C.: The secure shell (SSH) protocol architecture. IETF RFC 4251 (Standards Track) (January 2006)
14. Ylönen, T., Lonvick, C.: The secure shell (SSH) transport layer protocol. IETF RFC 4253 (Standards Track) (January 2006)
15. Dierks, T., Rescorla, E.: The transport layer security (TLS) protocol version 1.2. IETF RFC 5246 (Standards Track) (August 2008)
16. Kent, S., Seo, K.: Security architecture for the internet protocol. IETF RFC 4301 (Standards Track) (December 2005)
17. Kent, S.: IP authentication header. IETF RFC 4302 (Standards Track) (December 2005)
18. Kent, S.: IP encapsulating security payload (ESP). IETF RFC 4303 (Standards Track) (December 2005)
19. Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., Zorn, G.: Point-to-point tunneling protocol (PPTP). IETF RFC 2637 (July 1999)
20. IEEE: IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. amendment 6: Medium access control (MAC) security enhancements (July 2004)
21. Bleichenbacher, D.: Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In Krawczyk, H., ed.: CRYPTO '98. Volume 1462 of LNCS., Springer (1998) 1–12
22. Vaudenay, S.: Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS ... In Knudsen, L.R., ed.: EUROCRYPT 2002. Volume 2332 of LNCS., Springer (2002) 534–546
23. AlFardan, N.J., Paterson, K.G.: Lucky thirteen: Breaking the TLS and DTLS record protocols. In: IEEE Symposium on Security and Privacy 2013. (2013) To Appear.
24. Bellare, M., Canetti, R., Krawczyk, H.: Message authentication using hash functions - the HMAC construction. CryptoBytes **2**(1) (1996)
25. NIST: Advanced Encryption Standard (AES). Federal Information Processing Standards 197 (2001)

26. Dworkin, M.: Recommendation for block cipher modes of operation. Special Publication 800-38A (December 2001)
27. Rivest, R.: The RC4 encryption algorithm (March 1992)
28. NIST: Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC. NIST Special Publication 800-38D (2007)
29. Whiting, D., Housley, R., Ferguson, N.: Counter with CBC-MAC (CCM). IETF RFC 3610 (September 2003)
30. NIST: Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-4 (March 2012)
31. Simon, D., Aboba, B., Hurst, R.: The EAP-TLS authentication protocol. IETF RFC 5216 (March 2008)
32. UKPA: Acquirers' interface requirements for electronic data capture terminals. UKPA / APACS Standard 40, incorporated into Standard 70 Book 2, 4 & 5 (2007)
33. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: A block-cipher mode of operation for efficient authenticated encryption. In Reiter, M.K., Samarati, P., eds.: CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security, ACM (2001) 196–205
34. Rogaway, P., Bellare, M., Black, J.: OCB: A block-cipher mode of operation for efficient authenticated encryption. ACM Transactions on Information and System Security (TISSEC) **6**(3) (August 2003) 365–403
35. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In Miri, A., Vaudenay, S., eds.: SAC 2011. Volume 7118 of LNCS., Springer (2011) 320–337
36. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based encryption, authentication and authenticated encryption. In: DIAC 2012. (2012) `http://keccak.noekeon.org/KeccakDIAC2012.pdf`.
37. Saarinen, M.J.O.: Developing a grey hat C2 and RAT for APT security training and assessment. In: GreHack 2013 Hacking Conference, 15 November 2013, Grenoble, France. (2013) To Appear.
38. Bellovin, S.M.: Problem areas for the IP security protocols. In: Proc. Sixth USENIX Security Symposium. (1996) 205–214
39. Mitchell, J., Shmatikov, V., Stern, U.: Finite-state analysis of SSL 3.0. In: USENIX Security Symposium 1998, USENIX (1998) 201–216
40. Wagner, D., Schneier, B.: Analysis of the SSL 3.0 protocol. In: The Second USENIX Workshop on Electronic Commerce Proceedings, USENIX Press (November 1996) 29–40
41. Degabriele, J.P., Paterson, K.G.: Attacking the IPsec standards in encryption-only configurations. In: IEEE Symposium on Security and Privacy, IEEE Computer Society (2007) 335–349
42. Degabriele, J.P., Paterson, K.G.: On the (in)security of IPsec in MAC-then-encrypt configurations. In Al-Shaer, E., Keromytis, A.D., Shmatikov, V., eds.: ACM Conference on Computer and Communications Security, ACM (2010) 493–504
43. Paterson, K.G., Ristenpart, T., Shrimpton, T.: Tag size does matter: Attacks and proofs for the TLS record protocol. In Lee, D.H., Wang, X., eds.: ASIACRYPT 2011. Volume 7073 of LNCS., Springer (2011) 372–389
44. Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In Canetti, R., Garay, J.A., eds.: CRYPTO 2013, Part I. Volume 8042 of LNCS., Springer (2013) 429–448
45. International Standardization Organization: ISO/IEC 7816-4:2013 Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange. (2013)

46. International Standardization Organization: ISO/IEC 18000-63. Information technology – Radio frequency identification for item management – Part 6: Parameters for air interface communications at 860 MHz to 960 MHz Type C. (2012)
47. MODBUS: MODBUS Application Protocol Specification V1.1B. (April 2012) http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.
48. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. In: Ecrypt Hash Workshop 2007. (May 2007)
49. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sakura: a flexible coding for tree hashing. IACR ePrint 2013/213, http://eprint.iacr.org/2013/213 (April 2013)
50. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: On the security of the keyed sponge construction. In: SKEW 2011 Symmetric Key Encryption Workshop. (February 2011)
51. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge-based pseudo-random number generators. In Mangard, S., Standaert, F.X., eds.: CHES 2010. Volume 6225 of LNCS., Springer (2010) 33–47
52. Ferguson, N., Schneier, B.: Practical Cryptography. John Wiley & Sons (2003)
53. Saarinen, M.J.O.: CBEAM: Efficient authenticated encryption from feebly one-way phi functions. In: CT-RSA 2014: Cryptographers' Track, RSA Conference USA, 25–28 February 2014, San Francisco, USA, Springer (2014) To Appear.
54. Chang, S., R.Perlner, Burr, W.E., Turan, M.S., Kelsey, J.M., Paul, S., Bassham, L.E.: Third-round report of the SHA-3 cryptographic hash algorithm competition. Technical Report NISTIR 7896, National Institute of Standards and Technology (November 2012)
55. Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In Rogaway, P., ed.: CRYPTO 2011. Volume 6841 of LNCS., Springer (2011) 222–239
56. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. In Mangard, S., Standaert, F.X., eds.: CHES 2010. Volume 6225 of LNCS., Springer (2010) 1–15
57. Aumasson, J.P., Henzen, L., Meier, W., Naya-Plasencia, M.: Quark: A lightweight hash. Journal of Cryptology (2012) To Appear – Slightly Different Parameters from the CHES 2010 Version.
58. Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., Verbauwhede, I.: Spongent: A lightweight hash function. In Preneel, B., Takagi, T., eds.: CHES 2011. Volume 6917 of LNCS., Springer (2011) 312–325
59. Bernstein, D.: Curve25519: New Diffie-Hellman speed records. In Yung, M., Dodis, Y., Kiayias, A., Malkin, T., eds.: PKC 2006. Volume 3958 of LNCS., Springer (2006) 207–228